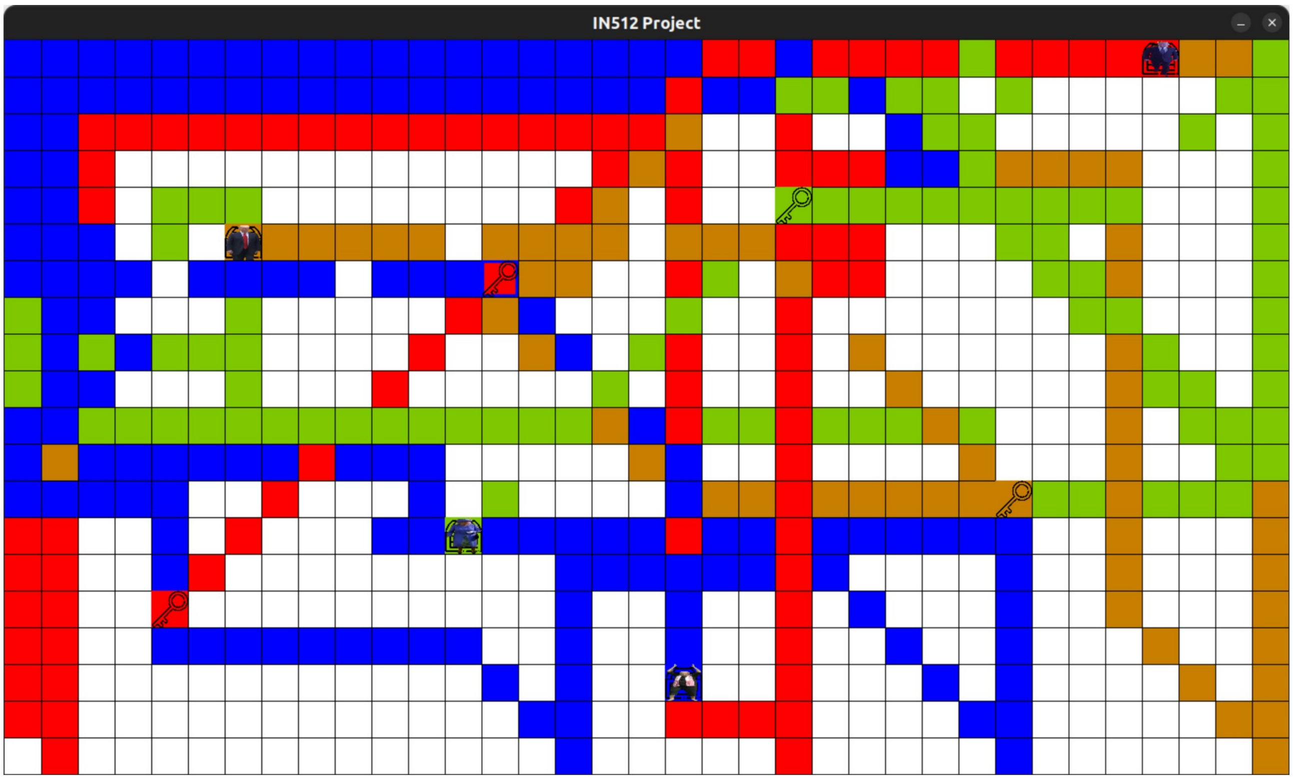
Nayel BLIDI
Léo DESCHAMPS
Clément POISSONNET

CDI

# Project In512
# Unlock Boxes



January 24th 2024

**Name : Strategy 2**

Strategy 2 is a purely algorithmic strategy that operates in three modes: exploration mode, prospection mode and reaching mode. For the first phase, robots locate all Points of Interest (PoI), during this phase they switch between exploration and prospection mode. When all PoI are located, robots enter in the second phase, the reaching mode. Each robot has his own map representation. If the value on its map is -1, then the cell is unexplored. In this case the robot will ask the game to retrieve the value of the cell and store it into its own map. We will now explain the goal of each mode:

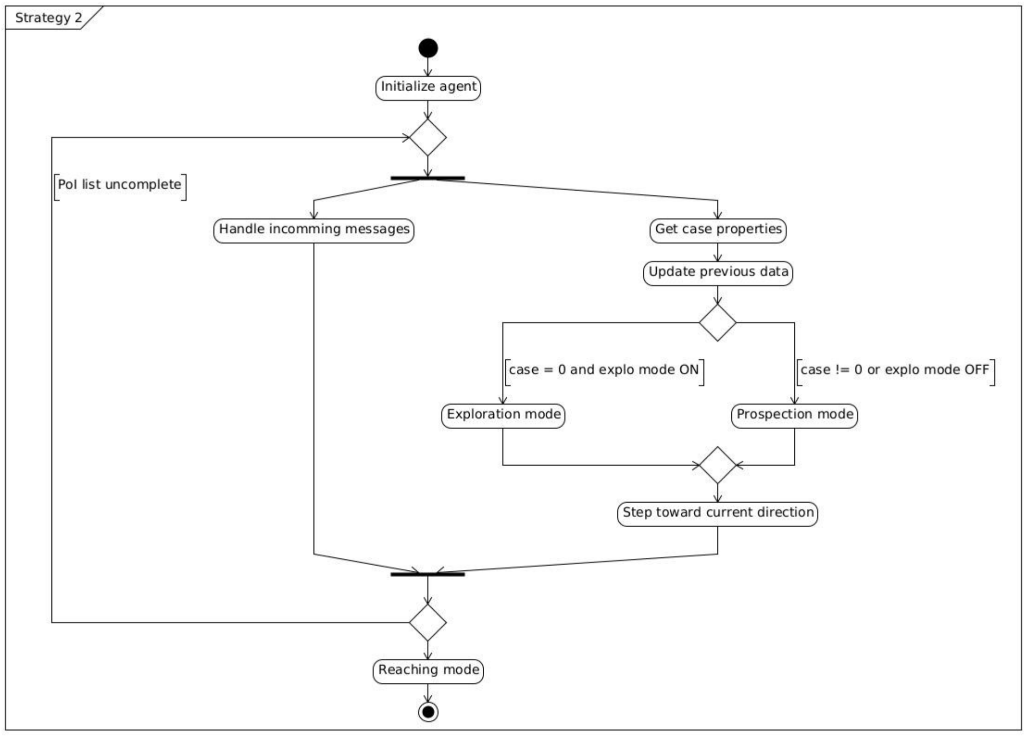
- **Exploration mode:** This is the default mode of the robot. When it is in this mode, the robot moves straight forwards. When the cell in front of him is already explored or if it is a wall, the robot will change its direction towards an unexplored cell (the eight cells around him). If all cells around him are explored, it keeps on moving forward even if there is a wall. In this case, it chooses a random direction to follow. While in this mode, the robot also check the value of the cell. If the current value is strictly greater than 0, it switches to prospection mode.

- **Prospection mode:** The goal of this mode is to locate a PoI. This mode is triggered when it encounters a value strictly greater than 0. To choose the direction, the robot checks if the value of the cell is increasing, decreasing or staying stable. If it is increasing, the robot continues forward. If it decreases, the robot turns back. Finally, if the value is stable, it means the direction is perpendicular to the PoI so it turns to the right. When the PoI is located, the robot saves data, sends it to other robots and considers all cells around PoI as explored. Then, it switches back to exploration mode.

- **Reaching mode:** Regularly, each robot checks if the PoI list is completed or not. If it is, robot switches to reaching mode. In this mode, the robot computes the shortest path from its position to the key and then to the chest. After that, the mission is accomplished.

To summarize all these ideas, the UML activity diagram below shows all the Strategy 2 process:

You can find in appendix the detailed UML activity diagram of Strategy 2



A strength  of this strategy is it never blocks itself. It always converges because when the robot seems to be blocked, it chooses a random direction.

**Limits and possible upgrades:**

- A first thing that we can see with Strategy 2 is that robots persist in exploring small parts of the map where obviously PoI cannot be. If there is a 5x5 area with explored zeros around it, and at least one explored value inside, then it means there won’t be any PoI in this area. We think this improvement is hard to add to our strategy, yet can greatly increase its efficiency.
- A second improvement that can be achieved is when a robot finds its own PoI, it should directly retrieve it without waiting for the reaching mode. This improvement can be easy to implement but it will not increase a lot the efficiency, especially if the number of robot is high.
- A third improvement that can be done is to use triangulation during prospection mode to locate the PoI faster. We think this improvement is hard to add to our strategy and can increase the efficiency slightly because of how small PoI’s areas are.

**Strategy 3 :**
Strategy 3 isn't a solution in itself, but the development of incremental propositions aiming at developing an AI that could solve the problem. Sadly, due to the definition of the game, (where the map evolves as the robot moves,) most of the trained AI find themselves soft locked, if not totally irrelevant. Considering the following models, here is a quick overview of what was the most challenging.

**- Dataset**:
Because the game is already defined, with fixed locations for the points of interest, existing runs can't be used as training examples. To overcome this limitation and still obtain a large number of features to train on, random maps of 20x20 cells are generated, with random 0s (explored places), -1s (unexplored), a random starting location (x_robot, y_robot), as well as a handful of points of interest (0.3, 0.5 and 1). The target is the best move, i.e. the direction that will reduce the most the distance between the current robot's position and the PoI with the highest interest value.

In the case of a flattened input, one batch becomes:
[x_robot, y_robot, flattended map (20x20=400)], [move]
This training example is thus a 1x402 values long vector, and "move" is the targeted label, i.e. the best direction to follow.

In the case of a 2D input, one batch becomes:

$$\begin{matrix} 0 or 1 & 0 or 1(1*20) & 0 or 1 \\ 0 or 1(20*1) & map(20*20) & 0 or 1(20*1) \\ 0 or 1 & 0 or 1(1*20) & 0 or 1 \end{matrix}$$

The input is the current agent map, padded with 0s, and 1s for cells that either correspond to the x_robot or y_robot positions. The target remains the same and is a unique integer value labeling the best direction to follow.

**- Fully connected architecture**
Usually, when it comes to AI solving games, the rules are well defined, and the AI must find a convincing path towards the objective. Similar problems involve a fixed map, with an obstacle and a target, and the player/robot must find a path towards the point of interest.
Such challenge would involve a well-known map, that is used as a constant training example input, with reinforced learning applied to the robot, whose effective inputs are solely its current (x, y) location, and output is the move label.

Considering a flattened input, we tried to feed enough training examples to the robot, so it can map the features in a way that would always encourage him to go towards the highest value PoI. Yet, this method is prompt to over-fitting, and although this can be balanced during training, the problem itself became an issue when testing the AI.

Because of the design of the game, the input doesn't change quickly. From one iteration to another, the robot has only traveled one cell, thus only slightly modifying three of the inputs of the AI: the (x,y) position, as well as the value of one cell of the map, most likely changing from -1 to 0. Because of this lack of differentiation, the input stays the same, and when computing the matrix products of the feed-forward evaluation, the output stays the same, thus soft locking the robot towards a single direction decision.

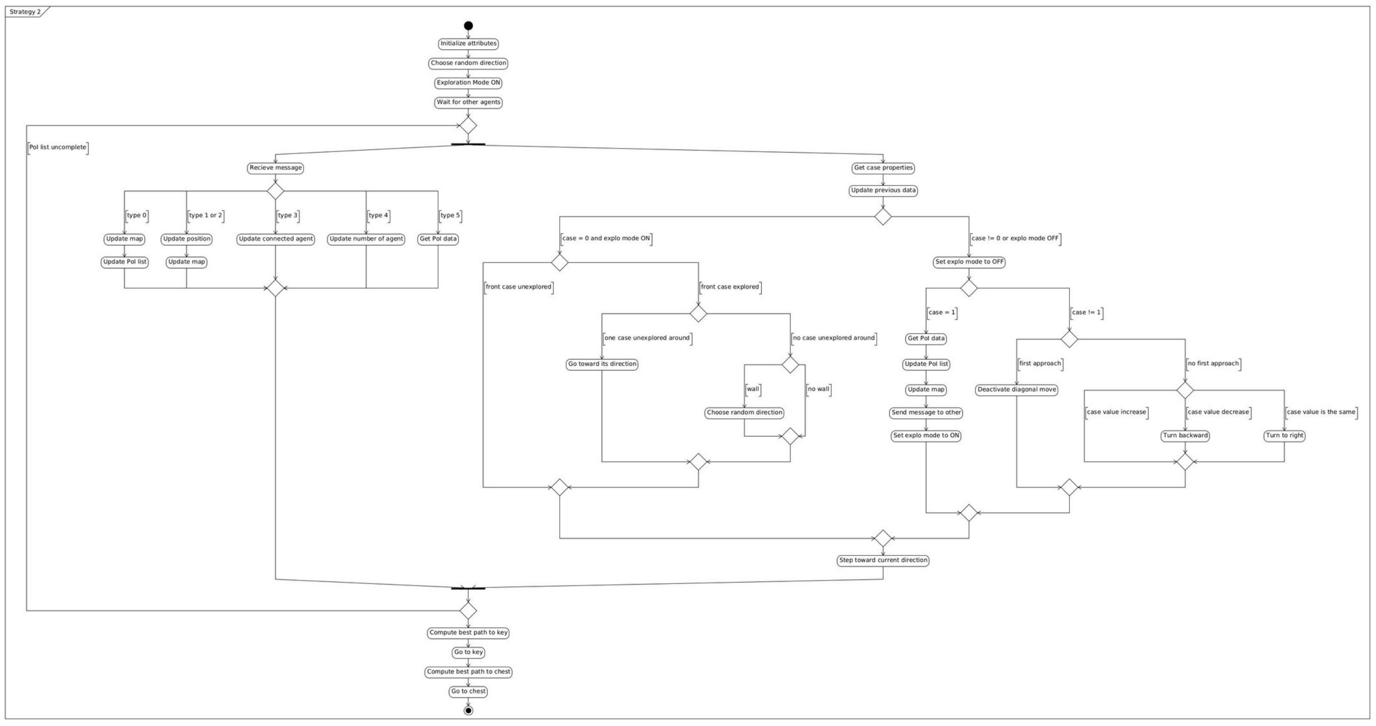
**- Convolutional Neural Network**
To overcome the aforementioned limitation, we’ve tried a more complex approach using convolution products, as well as a 2D input (as described in **Dataset**). This architecture is more likely to detect spatial features, but once again comes with a limitation, that is the impact of the position of the robot, as well as the importance of rare values (such as PoI).

This time, rather than getting locked, the robot may tend to move into different locations, but it completely fails at detecting the PoI. The model is either not trained enough (and randomly moves across the map), or over-fitted (and soft locks himself on a corner of the map).

**- Improvements**
Standard reinforced techniques are in either case not deep enough to train an AI to solve such a complex problem, due to the aspects of the evolving map (it is actually closer to a Go game rather than a labyrinth game).
Either deeper reinforced learning must be developed, with dedicated techniques, or the problem should be approached in a totally different way, such as only giving the adjacent values to the robot, or the list of previous moves etc…

## Conclusion:
The two strategies we have developed for this problem have totally different approaches. The first, purely arithmetical, seems to perform better, although there is room for improvement in certain respects. The second, based on a neural network, performs poorly given the complexity of the problem. We believe, however, that there is considerable room for improvement. The mix of these two strategies remains to be explored.

## Appendix



Detailed UML activity diagram for strategy 2